

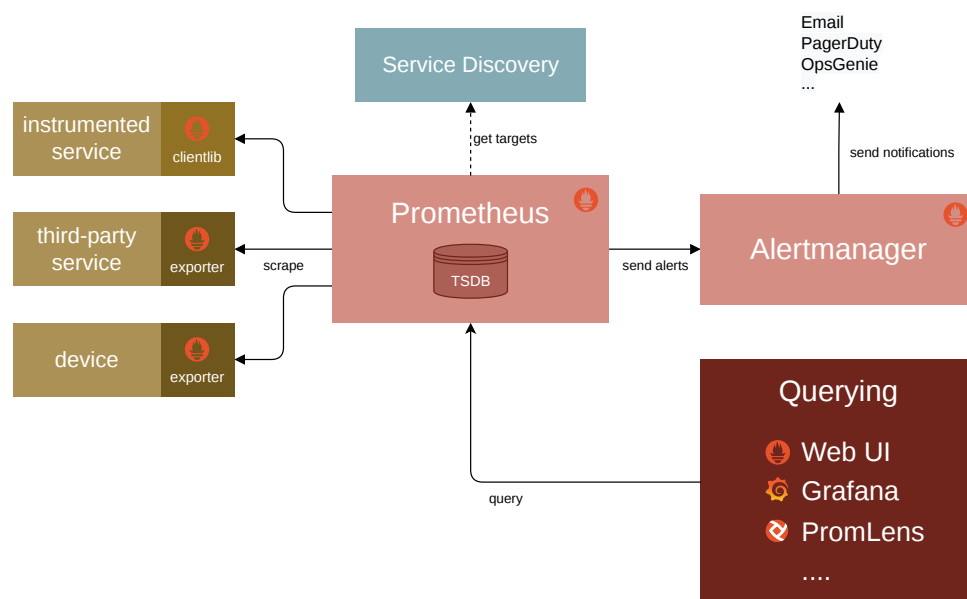


# Introduction to Prometheus

## 1. What is Prometheus?

- Prometheus is an open-source systems monitoring and alerting toolkit, optimised for reliability and performance in modern cloud-native and containerised environments (e.g., Kubernetes, Docker).
- It operates using a pull-based model where it periodically scrapes metrics from various endpoints that expose monitoring data via HTTP.

## 2. Prometheus' Ecosystem:



- **Prometheus Server:** Core component that collects and stores metrics.
- **Alertmanager:** Handles alerts based on the data collected, allowing you to define when thresholds are breached (e.g., high CPU usage).
- **Pushgateway:** Handles metrics from short-lived jobs or processes (such as batch jobs), which cannot be scraped easily.
- **Exporters:** Pre-built tools used to expose existing metrics from services, such as database metrics (e.g., MySQL, PostgreSQL).
  - **Pull-Based Model:** Prometheus scrapes targets at defined intervals by sending HTTP requests to defined endpoints. This makes it highly flexible and adaptable to dynamic infrastructure.

- **Time-Series Data Model:** Each data point is stored with a timestamp, a metric name, and key-value pairs called labels, making it highly efficient for querying.

## ▼ Key Features of Prometheus

### 1. Multi-Dimensional Data Model:

- Prometheus stores data as time-series with multiple dimensions (labels). This means a single metric can track multiple instances (e.g., multiple nodes, applications) with distinguishing labels like `job="app1"`, `instance="localhost:8080"`.
- Prometheus uses a key-value system for labels. For example, you can differentiate between HTTP requests based on the endpoint, status code, or method.

### 2. PromQL (Prometheus Query Language):

- A powerful, flexible query language used to query Prometheus metrics.
- PromQL allows for basic operations like summing, averaging, or filtering specific metrics based on labels. You can create complex queries for monitoring system health and detecting anomalies.
- This query retrieves the rate of HTTP 500 error requests in the last 5 minutes.

```
sum(rate(http_requests_total{status="500"}[5m]))
```

#### Querying basics | Prometheus

An open-source monitoring system with a dimensional data model, flexible query language, efficient time series database and modern alerting approach.

 <https://prometheus.io/docs/prometheus/latest/querying/basics/>



### 3. Service Discovery:

- Prometheus can automatically detect services to monitor, thanks to integration with Kubernetes, Consul, and other service discovery tools. This is critical in dynamic environments where services frequently scale up or down.

### 4. Alerting and Notification:

- **Alerting with `Alertmanager`:**
  - Alerts are generated based on Prometheus' collected metrics and are routed to `Alertmanager`. You can define alerting rules based on metric thresholds (e.g., CPU usage > 90% for more than 5 minutes).
- **Customizable Alert Routing:**
  - `Alertmanager` can be configured to send alerts through email, Slack, PagerDuty, etc., allowing teams to respond quickly to issues.

### 5. Long-Term Storage and Integration:

- Prometheus stores data on a local disk by default, with options for configuring retention periods (e.g., keeping data for 15 days).
- Prometheus can push older metrics to external storage systems for long-term storage, like Thanos or Cortex.

## ▼ Integrating Prometheus with Spring Boot

### Step 1: Adding Micrometer and Actuator Dependencies

- **Micrometer** is used to instrument Spring Boot applications, and **Spring Boot Actuator** exposes the metrics endpoint that Prometheus can scrape.
- Add the following dependencies to your `pom.xml` to enable Prometheus metrics:

```
<dependency>
  <groupId>io.micrometer</groupId>
  <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

## Step 2: Enabling the Prometheus Endpoint

- Spring Boot Actuator will expose a `/actuator/prometheus` endpoint, which Prometheus uses to collect metrics.
- Enable this endpoint by adding the following configuration to `application.properties`:

```
management.endpoints.web.exposure.include=prometheus
management.endpoint.prometheus.enabled=true
```

- This configuration ensures that the Prometheus endpoint is available for scraping.

## Step 3: Configuring Prometheus to Scrape Metrics from Spring Boot

- In your `prometheus.yml` file, configure Prometheus to scrape the `/actuator/prometheus` endpoint exposed by your Spring Boot app.
- Since we'll run Prometheus and the Spring Boot app in Docker, we need to set up the correct target. Here's the configuration:

```
scrape_configs:
  - job_name: 'spring-boot-app'
    scrape_interval: '5s' # Optional: Adjust frequency as needed
    metrics_path: '/actuator/prometheus' # Specify the metrics path
    static_configs:
      - targets: ['movie-service:8080'] # Replace with the container name or
        correct hostname
```

- **Note:** Adjust the target to match your environment. If running in Docker with a network, use the container name (e.g., `movie-service`), or use `host.docker.internal` if Prometheus is on Docker and Spring Boot is running locally (for Mac/Windows).

## Step 4: Running Prometheus and Spring Boot in Docker

### 1. Set Up a Docker Network (if running both in Docker):

- This allows the Prometheus and Spring Boot containers to communicate.

```
docker network create monitoring
```

### 2. Run Spring Boot Application in Docker:

- Build a Docker image for your Spring Boot application (if you haven't already) and run it on the `monitoring` network.

```
docker build -t movie-service .
docker run -d --name movie-service --network monitoring -p 8080:8080 movie-s
ervic
```

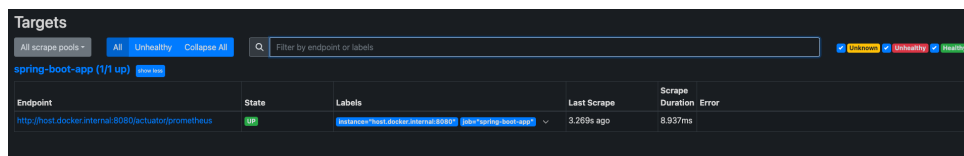
### 3. Run Prometheus in Docker with Configurations:

- Start Prometheus on the same network with the `prometheus.yml` configuration file mounted.

```
docker run -d --name prometheus --network monitoring -p 9090:9090 -v $(pwd)/
prometheus.yml:/etc/prometheus/prometheus.yml prom/prometheus
```

### 4. Verify Setup:

- Visit `http://localhost:9090/targets` to check the Prometheus **Targets** page. You should see `spring-boot-app` listed as `UP` if everything is configured correctly.
- This confirms that Prometheus is successfully scraping metrics from your Spring Boot application.



Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://host.docker.internal:8080/actuator/prometheus	UP	instance="host.docker.internal:8080" job="spring-boot-app"	3.269s ago	8.937ms	

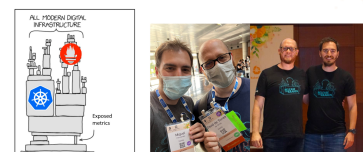
## ▼ ⚠ But, watch out !

### How attackers use exposed Prometheus server to exploit Kubernetes clusters

Kubernetes and Prometheus advise problems with exposing your data to the world, but regardless, exposed Prometheus are still widespread.

<https://sysdig.com/blog/exposed-prometheus-exploit-kubernetes-kubeconeu/>

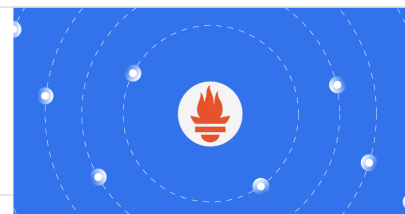
### Kubernetes fingerprinting with Prometheus



### Hacking your way to Observability—Part 1

Have you ever seen someone trying to find how to solve an issue and a dozen people watching or suggesting to him/her what to look for as if they were trying to solve a mystery? . These situations...

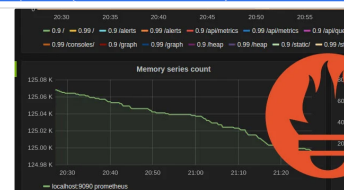
<https://jonbc.medium.com/hacking-your-way-to-observability-part-1-cf4cd42fb4dc>



### Experts Warn of Unprotected Prometheus Endpoints Exposing Sensitive Information

Security experts warn of unprotected Prometheus endpoints that can expose sensitive data.

<https://thehackernews.com/2021/10/experts-warn-of-unprotected-prometheus.html>



## ▼ Security Considerations for Prometheus Setup

While Prometheus is a powerful tool for monitoring, it's essential to secure your setup to protect sensitive data and metrics. Here are some recommended practices for securing Prometheus, especially when exposing endpoints on networks:

### • Securing the Prometheus Web UI with Authentication

By default, Prometheus doesn't offer built-in authentication for accessing the web UI. Here's how you can add basic authentication:

- **Use a Reverse Proxy:** Set up a reverse proxy (such as NGINX or Apache) in front of Prometheus to add authentication.

- **Example with NGINX:**

1. Install NGINX and configure a reverse proxy.
2. Create an authentication file:

```
sudo htpasswd -c /etc/nginx/.htpasswd <username>
```

3. Configure NGINX to require authentication for Prometheus:

```
server {
    listen 9090;
    location / {
        proxy_pass http://localhost:9090;
        auth_basic "Prometheus Authentication";
        auth_basic_user_file /etc/nginx/.htpasswd;
    }
}
```

4. Restart NGINX and access Prometheus via `http://your-nginx-server:9090`.

- This setup ensures only authenticated users can access the Prometheus interface.

- **Limiting Access to the Prometheus Web UI**

If Prometheus is deployed in a production environment, consider restricting access to the Prometheus web UI to a specific network or secure it behind a VPN:

- **IP Whitelisting:** Configure your network firewall to allow only specific IP addresses (e.g., your internal network or VPN IPs) to access the Prometheus server.
- **VPN Access:** If you're running Prometheus on a cloud provider, consider setting up a VPN to allow only authorised users within your organisation to access it.

- **Securing the `/actuator/prometheus` Endpoint in Spring Boot**

Since the `/actuator/prometheus` endpoint exposes detailed application metrics, it's best to limit access to this endpoint as well:

- **Restrict Endpoint Exposure:** Ensure the endpoint is exposed only on internal networks. Avoid making it publicly accessible if possible.
- **Spring Security for Authentication:**
  - If you're using Spring Security, you can add basic authentication to the `/actuator/prometheus` endpoint:

```
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration
@EnableWebSecurity
```

```
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers("/actuator/prometheus").authenticated()
                .and()
            .httpBasic();
    }
}
```

- You can also configure usernames and passwords in `application.properties`:

```
spring.security.user.name=prometheus_user
spring.security.user.password=your_secure_password
```

- **Enable HTTPS for Prometheus and Spring Boot Endpoints**

- **Prometheus with HTTPS:** If Prometheus is exposed over a network, consider using a reverse proxy (like NGINX) with SSL to serve the Prometheus UI over HTTPS.
- **Spring Boot with HTTPS:** You can enable HTTPS in Spring Boot by adding a keystore and configuring `application.properties`:

```
server.port=8443
server.ssl.enabled=true
server.ssl.key-store=classpath:keystore.p12
server.ssl.key-store-password=your_password
server.ssl.key-store-type=PKCS12
server.ssl.key-alias=your_alias
```

- **Restrict Access to Exporters (e.g., Node Exporter)**

Exporters like Node Exporter are often deployed on server nodes and expose detailed system metrics. Ensure they are accessible only from trusted sources:

- **Firewall Rules:** Configure your firewall to limit access to the exporter's port (e.g., `9100` for Node Exporter) so that only Prometheus can scrape these metrics.

- **Disable Unused or Sensitive Endpoints**

In production, it's best practice to disable any unused Actuator endpoints or limit them to internal networks:

- In `application.properties`, specify which Actuator endpoints to expose:


```
management.endpoints.web.exposure.include=health,info,prometheus
```

- This configuration restricts access to only the `health`, `info`, and `prometheus` endpoints, hiding other potentially sensitive endpoints from exposure.

## ▼ Further Resources

## GitHub - prometheus/docs: Prometheus documentation: content and static site generator

Prometheus documentation: content and static site generator - prometheus/docs

 <https://github.com/prometheus/docs>


## prometheus/docs

Prometheus documentation: content and static site generator

 583 Contributors  131 Issues  4 Discussions  663 Stars

## Quick Guide to Micrometer | Baeldung


Learn about the metrics facade Micrometer and its integration with Spring.

 <https://www.baeldung.com/micrometer>



## PromQL Cheat Sheet

PromLabs - We teach Prometheus-based monitoring and observability

 <https://promlabs.com/promql-cheat-sheet/>